AIPI 520 Project 1

Andrew Jin, Diya Mirji, Danny Ross

The objective of this project is to predict the hourly temperature at Raleigh-Durham International Airport (RDU) during the period September 17, 2025 12am - September 30, 2025 11pm.

Data

Our data for the temperature at RDU was sourced from Meteostat's bulk data where the station ID for RDU is 72306. The .csv file for each year contained columns with the date, hour, weather condition code, temperature, dew point, sunshine duration, precipitation, snow depth, wind direction, wind speed, peak gust, air pressure, and relative humidity. Since we are not allowed to use other features than the date to generate our predictions, we only used the year, month, day, hour, and temperature columns in our training. We took the year, month, day, and hour columns to create the Datetime index. We exported data from the years 2020 to 2025. So our dataset consisted of consecutive hourly temperatures from January 1, 2020 to September 30, 2025. We used the last 5 years of data because we wanted our model to see the temperature from during the same time in the previous years. This data was then split into a test and training dataset, where the test set contained the hourly temperature from September 17, 2025 to September 30, 2025 and the rest was used as the training data.

We conducted an Exploratory Data Analysis on the datasets. First, we verified that there were no missing temperature values. Next, we checked the distribution of the temperature data. Since the mean and the 50% quartile were almost the same value and the skewness was slightly negative, we found that the data had a roughly symmetric distribution with a slight left skew. The box plots also showed a few outliers, but we chose to keep those samples as the distribution was mostly symmetric. Then, we explored the seasonal decomposition of the temperature from the previous year, September 17, 2024 to September 30, 2024. There was a non-linear trend, but a strong seasonality every day. From there, we checked the autocorrelation plots for an hourly, daily, and monthly lag. There was a strong positive correlation with every previous hour in a day that passed the 95% confidence interval, especially the 1 hour of lag and 24 hours of lag. The daily lag autocorrelation plot showed a strong positive correlation with every previous day in a month above the confidence interval, especially with 1 previous day. There was a strong positive correlation with a 1 month lag and a strong negative correlation with a 6 month lag. Based on these findings with the autocorrelation plot, we can use the lags as new features. Lastly, we wanted to check if there are any similar trends with the same time period in the different years. While there was some daily seasonality, there were no obvious patterns of trends between 2020-2024.

Additionally, we were also interested in using cities with similar climate to that in Raleigh to see if it would generate similar predictions. Several sources stated that Kyoto, Japan had a similar climate so we also used the Meteostat data with unique ID 47759 for temperature data from Kyoto.

Linear Regression Model

We initially only extracted data from the date range September 17 - September 30 for each year (2020-2025) and used it as our training data. After separating the date in different columns, we experimented by using different combinations of features to see if any parts of the date were more relevant in creating the predictions. This method overall yielded poor results and a negative R^2 value so we decided to explore other aspects of the time series model. Instead of only using the same date range for each year, we decided to use all consecutive dates and their temperature from 2020-2025 to give the model more features in an attempt to create a more robust linear regression model. We hypothesized that the temperature in a given hour would be somehow correlated to the values in the previous hours and days. Thus, we implemented autocorrelation to see if this improved the performance. Based on our exploratory data analysis, there seemed to be a strong correlation of the temperature to the previous hour, day, and year so we built a new model by adding new columns to the dataframe that contained the temperature from the previous hour, day, and year.

We also added pure calendar features - hour, day, month, year, day of the year, day of the week, sine and cosine of the hour, sine and cosine of the day of the year, and sine and cosine of the month cyclical encodings which transform our calendar features using sin and cos transforms and lag features. The cyclical encodings of hour, day of the year, and month allows the model to learn the cycle of the temperature during these time periods. For example, since the temperature is warmest during the day around hour 12 to 14 and the temperature is coolest during the night from hour 22 to 3, we want the model to learn a smooth oscillating pattern and to recognize that hour 23 and hour 0 are consecutive hours. We also tried to capture any cycles that have 2 or 3 peaks of weather changes, such as bi/tri-annual rainy seasons, through harmonics of hour and day of the year cycles.

After training our Linear Regression model with the new features, we analyzed the coefficients of the Linear Regression equation to find the importance of each feature to the model. The larger the magnitude of the coefficient is, means the greater the feature's importance is to the model. The largest contributors to the model were: the temperature from the previous hour (w=0.972), the cosine of the hour (w=-0.821), and the sine of the hour (w=-0.791). Since we had many added features to the dataset, we ran Ridge Regression to the dataset to see if there were

any noisy or unnecessary features. After increasing and decreasing the alpha value for Ridge regularization, we found that the performance did not change. Therefore, we chose not to remove any of the features.

Building our model on autocorrelation significantly improved our results where it improved the mean square error (MSE), mean absolute error (MAE), the R^2 score from 22.76, 3.78, -0.18 to 1.11, 0.82, 0.94 respectively. We use MSE because this metric penalizes large errors so we want to see if our incorrect predictions are off by a significant amount. Since we are using autocorrelation, being off by one time can be worse as it will affect the consecutive predictions. MAE is more robust to outliers and is easier to interpret especially if we were to show this to others. The R^2 value gives us an overview of how much variability the model explains. These metrics altogether give us a detailed overview of our model's performance and tell us that our linear regression model does a good job of predicting the temperature with minimal errors.

We attempted to apply the same autocorrelation techniques to the dataset containing temperatures in Kyoto, Japan. We also applied the same training techniques as we did on the RDU weather and compared the predictions. The MSE, MAE, and R^2 were 43.95, 5.58, and -1.28 respectively. From the metrics, we observe that training on the model on Kyoto's past temperatures doesn't do a good job of predicting the temperature at RDU despite the similarities in climate. Since autoregression models depend on past values, using temperatures in a different region could lead the predictions in the wrong direction. The temperatures leading up to September 17 in Kyoto are most likely very different compared to that of RDU, which is why training the model on Kyoto data didn't yield desirable results.

Random Forest Model

Although the linear regression model works well, in order to improve upon it and capture non-linear dynamics in the data, we decided to use a more complex model - random forest. The random forest model uses multiple decision trees in order to model complex interactions and autoregression. One of the key advantages of random forest over linear regression is its ability to automatically capture non-linear relationships and interactions between features without us having to manually specify them. This is particularly useful for weather data where relationships between variables can be complex and context-dependent.

The feature engineering process built upon what we used for the linear model. We kept the same calendar features—hour, day, day of week, month, year—along with the cyclical and harmonic encodings using sine and cosine transformations. These cyclical encodings are important because they properly represent the circular nature of time. For example, hour 23 and hour 0 are actually adjacent in time, but numerically they look far apart. The sine and

cosine transformation fixes this problem, and we used multiple harmonics to capture more complex patterns throughout the day and year. The linear regression model used basic lag features from the previous hour and previous day. For the random forest model, we significantly expanded this by including lag values at 1, 3, 6, 12, 24, and 48 hours to give the model more flexibility in learning from different time scales. More importantly, we added rolling statistics where 24-hour rolling means and standard deviations to capture short-term trends and volatility, plus a 168-hour (weekly) rolling mean to capture longer-term weather patterns. These rolling features aggregate information over windows of time rather than just single point estimates, giving the model a better understanding of recent temperature dynamics. An important consideration during feature engineering was avoiding data leakage. For training data, we only used training data to create the lags. For test data, the lag features look back at historical values including the end of training data, which is fine because in a real forecasting scenario we'd have access to all historical data up to the prediction point. The key is that training data never looks forward into future test values.

Once these features were properly developed, we built our random forest model using scikit-learn within a pipeline which handles NaN values through median imputation. These NaN values occur at the very beginning of our time series where we don't have enough history to calculate all the lag features. For hyperparameter tuning we used TimeSeriesSplit with 5 folds, which is a time-aware cross validation strategy. Unlike regular cross-validation which randomly splits data, TimeSeriesSplit respects temporal ordering by always training on earlier data and validating on later data, which mimics how we'd actually use the model in practice.
The hyperparameters tuned during this process are as follows:

- Number of trees: n_estimators = [100, 300]
- Maximum depth: max_depth = [10, 12] - controls how deep each tree can grow
- Min samples per leaf: min_samples_leaf = [1, 2] - provides regularization to prevent overfitting

This hyperparameter search evaluated 18 different configurations using negative mean absolute error as our scoring metric. The final hyperparameters from this process were n_estimators = 100, max_depth = 10, min_samples_leaf = 1. After hyperparameter tuning, we trained the best model on the complete training dataset and evaluated its performance on the held-out test set from September 17th to September 30th, 2025. The model achieved the MSE, MAE, R^2 Score 0.756, 0.952, 0.95 respectively. Feature importance analysis showed that the lag features, particularly the 1-hour lags, were clearly the most important predictor, which makes sense given the strong autocorrelation we saw in temperature data. The rolling statistics and cyclical encodings also contributed. The model was able to capture both the short-term hour-to-hour variations and the longer-term multi-day weather patterns in our test period.